پروژه هشتم علوم اعصاب محاسباتی

- در فازهای قبلی، تاثیر پارامترهای رمزنگاری به تفصیل مورد بررسی قرار گرفته‌اند. از همین رو، با توجه به حجم زیاد نمودارهای این تمرین، از بررسی این پارامترها صرف نظر کرده و به بررسی دقیق فیلترها می‌پردازیم.
- کدها فقط در ابتدای گزارش زیاد هستند.
- هر بخش با تعداد زیادی نمودار که به ترتیب با هدف مقایسه چیده شده‌اند شروع خواهد شد و پس از نمودارها، درمورد پارامتر مربوطه تحلیل انجام خواهد شد.
- آزمایش‌ها با اندازه کرنل‌های بزرگ انجام شده‌اند تا نمایش بصری بهتری داشته باشند.

0. فهرست مطالب

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
import torch
pi = torch.acos(torch.zeros(1)).item() * 2
path1 = "image1.jpg"
```

برای پرهیز از تکرار نوشتن، تابع شبیه‌سازی لازم برای این تمرین تعریف شده است. تمام پارامترها قابل تغییرند.

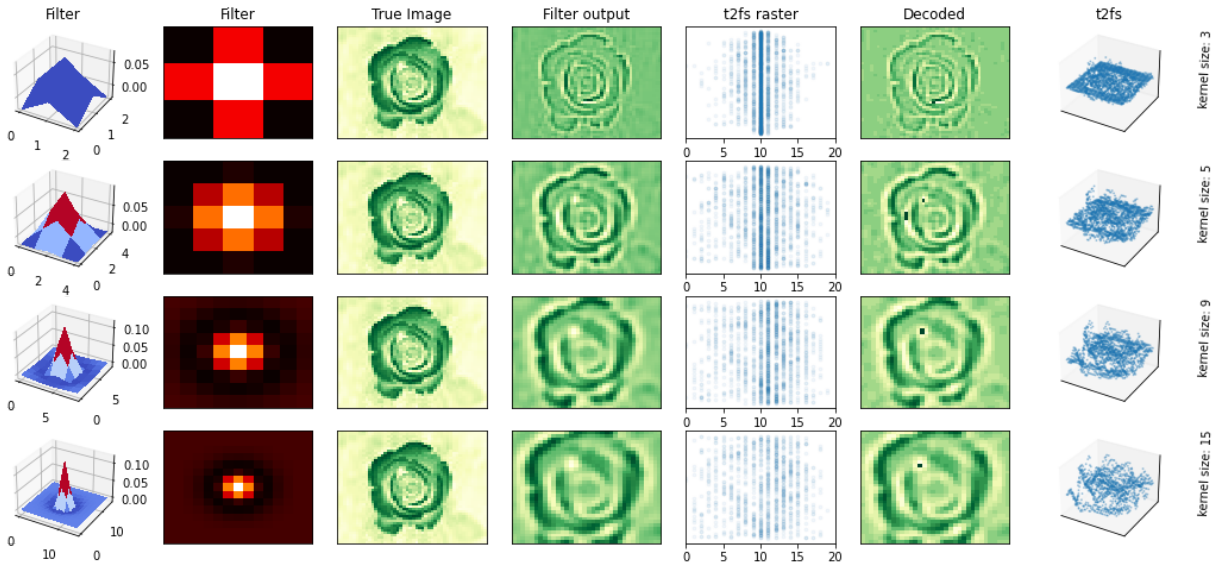
In [3]:
```
from cnsproject.network.filters import DoGFilter,GaborFilter
from cnsproject.network.kernels import DoG_kernel,gabor_kernel
from torchvision import transforms
from cnsproject.network.encoders import *
from cnsproject.monitors.monitors import Monitor
from cnsproject.monitors.plotter import Plotter
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib import image
from PIL import Image
import numpy as np

time = 20
dt=1
```

```
def simulate(p, encoder='t2fs', time=time, name='', postfix='', filter_name='
             title=True, **args):
    if filter_name=='DoG':
        Filter = DoGFilter
        Kernel = DoG_kernel
    else:
        Filter = GaborFilter
        Kernel = gabor_kernel
    kernel_data = Kernel(**args)[0][0]
    p.surface_3d('filter'+postfix+'3D', title='Filter' if title else '',
                 data={'z': kernel_data}, cmap=cm.coolwarm, antialiased=False

    p.imshow('heat_filter'+postfix, kernel_data, cmap='hot', interpolation='n

    im = np.array(Image.open(path1).convert('L'))
    p.imshow('true_image'+postfix, im, title="True Image" if title else '', c

    filt = Filter(transform=transform, **args)
    filter_output = filt(im)[0][0].numpy()
    filter_output -= filter_output.min()
    filter_output /= filter_output.max()
    filter_output *= 255
    p.imshow('filter_output'+postfix, filter_output, title="Filter output" if
             cmap='YlGn', interpolation='nearest')

    if encoder=='t2fs':
        enc = Time2FirstSpikeEncoder(name='enc', shape=filter_output.shape, m
    else:
        enc = PoissonEncoder(name='enc', shape=filter_output.shape, max_input
    enc.encode(torch.from_numpy(filter_output))
    enc_monitor = Monitor(enc, state_variables=["s"], time=time, dt=dt)
    enc_monitor.reset()
    enc_monitor.simulate(enc.forward, {})
    p.population_activity_raster('raster'+postfix, monitor=enc_monitor, y_lab
                                 s=7, alpha=.05, y_vis=False, title=encoder+'

    p.imshow('decode'+postfix, enc.decode(enc_monitor['s']), cmap='YlGn', int
             title="Decoded" if title else '')

    p.population_activity_3d_raster('raster'+postfix+'3D', monitor=enc_monitc
            z_vis=False, x_vis=False, y_vis=False, z_r=True, title=encoder if
    return enc_monitor
```

1. فیلتر DoG

A.1. اثر اندازه کرنل

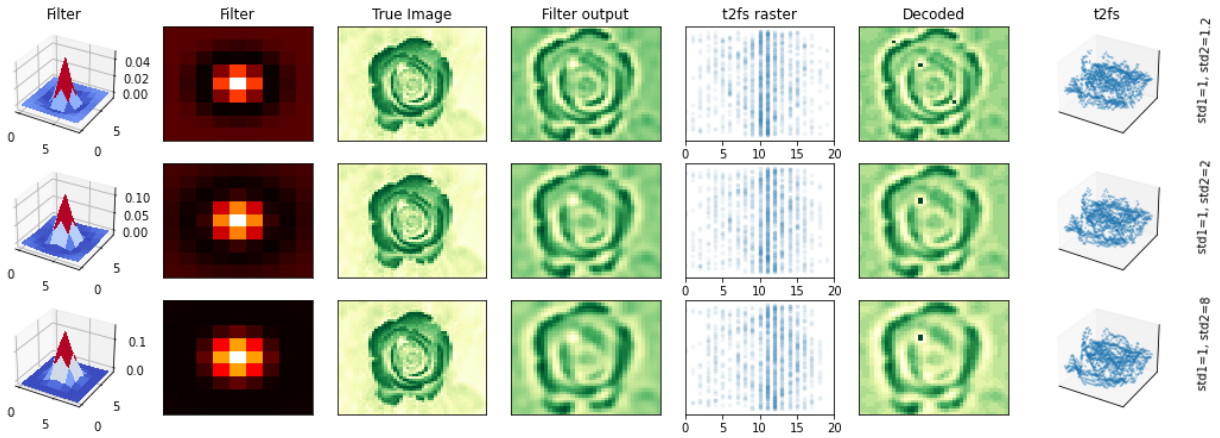
In [43]:
```
i_max = 4
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  kernel_size = [3,5,9,15][i]
  simulate(p, encoder='t2fs', name=f'kernel size: {kernel_size}', filter_name
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=kernel_size, std1=1., std2=2.)
p.show()
```



مشاهده می‌کنیم که با افزایش اندازه کرنل، شفافیت تصویر حاصل از کرنل کاهش پیدا می‌کند، در شکل فیلتر فرورفتگی قوی‌تری پدیدار می‌شود، تراکم اسپایک‌ها در خروجی رمزنگاری کاسته می‌شود و این اسپایک‌ها دارای بی‌نظمی بیشتری هستند. دلیل شفافیت کمتر واضح است، چون پیکسل‌های بیشتری با هم مورد بررسی قرار می‌گیرند، هر پیکسل سهم کمتری در خروجی ایفا می‌کند. دلیل فرورفتگی بیشتر در شکل فیلتر مستقیما به فرمول مورد استفاده وابسته است. دلیل تراکم کمتر در خروجی رمزنگار نیز آن است که الگوی مورد بررسی در کرنل با ابعاد بزرگ‌تر کمیاب‌تر بوده و درنتیجه فرکانس اسپایک کاهش پیدا می‌کند. بی‌نظمی بیشتر نیز دلالت بر آن دارد که بخش‌های ساده تصویر نادیده گرفته می‌شوند.

B.1. اثر اختلاف انحراف معیار دو توزیع

In [51]:
```
i_max = 3
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  std2 = [1.2,2,8][i]
  simulate(p, encoder='t2fs', name=f'std1=1, std2={std2}', filter_name='DoG',
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=9, std1=1., std2=std2)
p.show()
```

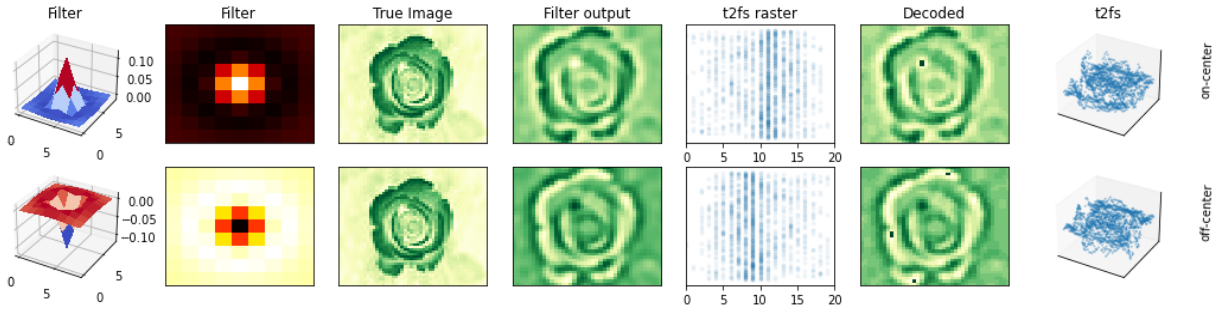


همانطور که دیده می‌شود، با افزایش نسبت دو احراف معیار توزیع‌ها، فرورفتگی کمتر می‌شود و درنتیجه قدرت مدل در تشخیص نقاط روشن در زمینه تاریک کاسته شده و کیفیت تصویر بازیافتی کم می‌شود.

این پدیده با توجه به فرمول و منطق فیلتر مورد استفاده، مورد انتظار بود.

C. Off-Center.1

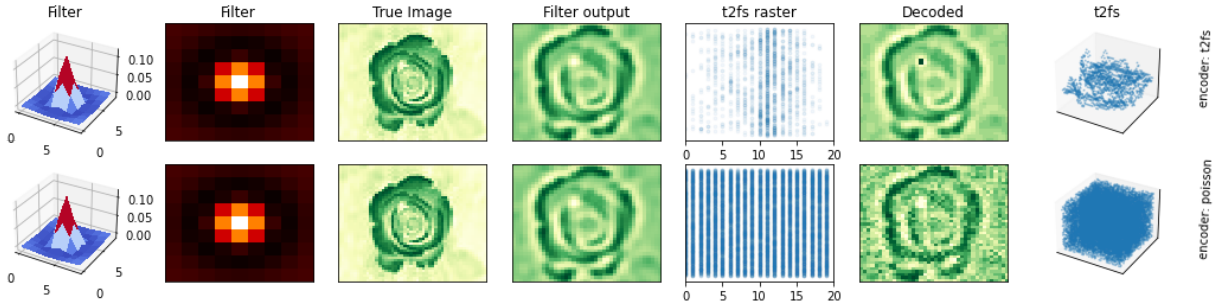
In [54]:
```
i_max = 2
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  simulate(p, encoder='t2fs', name=['on-center','off-center'][i], filter_name
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=9, std1=1., std2=2., off_center=i==1)
p.show()
```



به سادگی تئوری مطرح شده در مباحث درس برای این قسمت، در نمودارها قابل مشاهده است. در فیلتر on-center، نقاط روشن در زمینه تاریک مورد توجه قرار گرفته و روشن‌تر شده‌اند و در فیلتر off-center برعکس. برای صحت‌سنجی این صحبت، به مرکز گل توجه کنید.

D.1. تعامل با رمزنگار time to first spike و پواسون

In [55]:
```
i_max = 2
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  enc = ['t2fs','poisson'][i]
  simulate(p, encoder=enc, name=f'encoder: {enc}', filter_name='DoG', postfix
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=9, std1=1., std2=2.)
p.show()
```



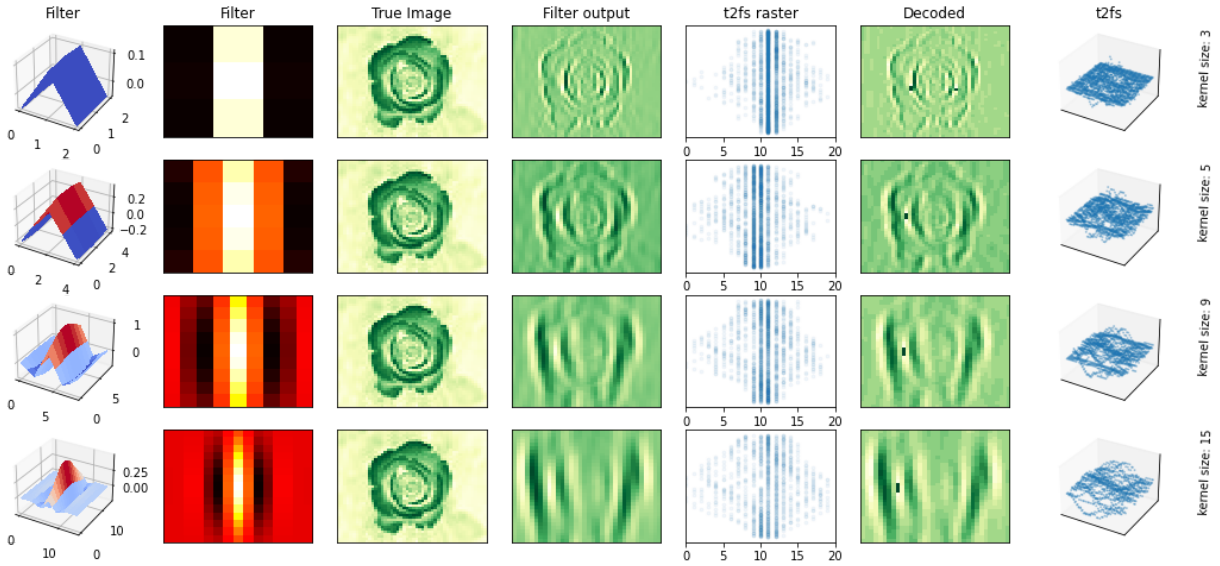
پیش‌تر نیز به اثر فیلترها بر روند کدگذاری اشاره شد. با فیلتر کردن بخشی از دادگان تصویر حذف می‌شود (فقط نقاطی باقی می‌مانند که با فیلتر همخوانی داشته باشند) درنتیجه خروجی رمزنگار تراکم

کمتری خواهد داشت و پس از کدگشایی، به تصویر پس از فیلتر تبدیل خواهد شد و توانایی بازسازی تصویر اصلی را ندارد. تفاوت دو فیلتر در اینجا نیز همانیست که در فازهای قبلی بیان شده است.

2. فیلتر Gabor

A.2. اثر اندازه کرنل

In [42]:
```
i_max = 4
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  kernel_size = [3,5,9,15][i]
  simulate(p, encoder='t2fs', name=f'kernel size: {kernel_size}', filter_name
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=kernel_size, wavelength=5, std=2, orientation=torch.tensor(0.
p.show()
```

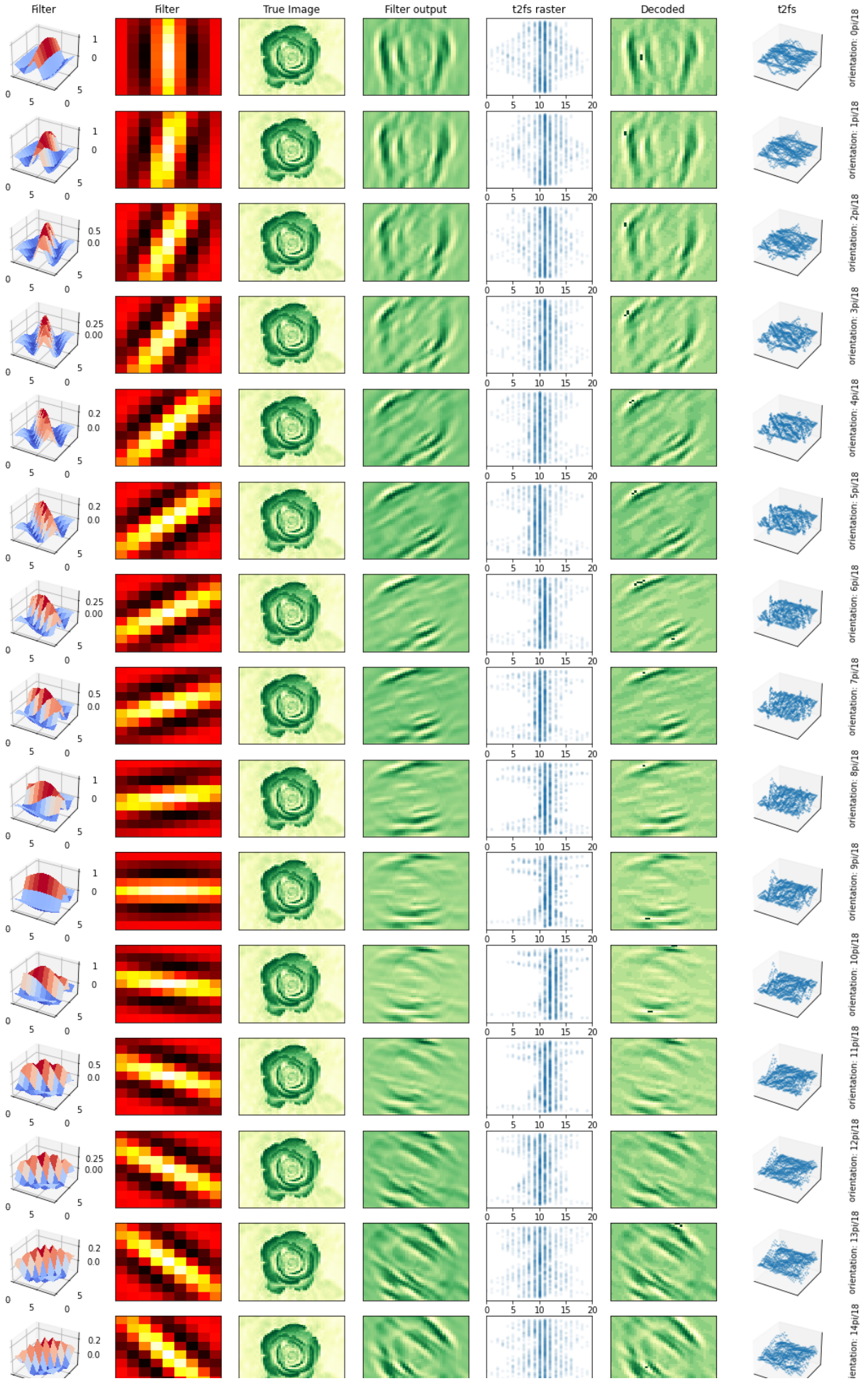


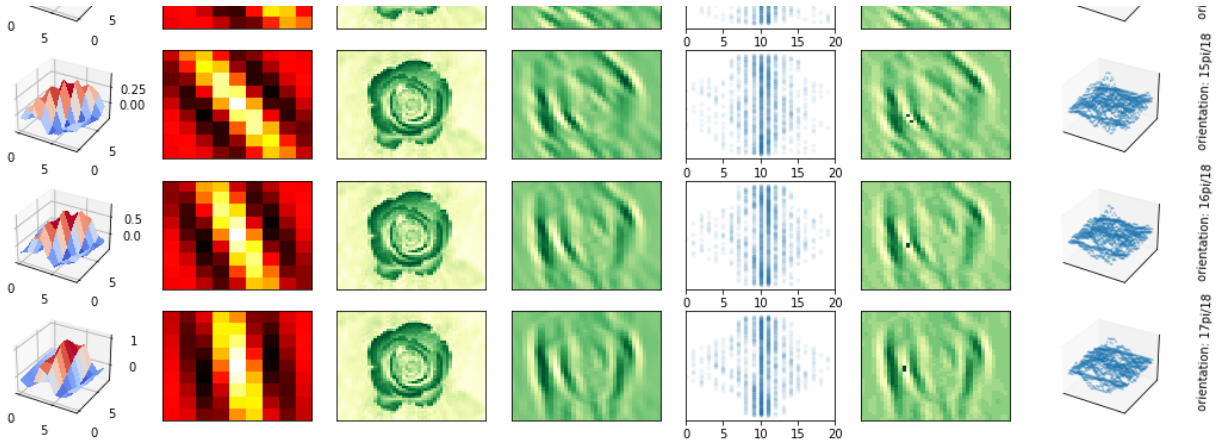
مشاهده می‌کنیم که با افزایش اندازه کرنل، شفافیت تصویر حاصل از کرنل کاهش پیدا می‌کند، در شکل فیلتر موج‌های دوم و سوم نیز پدیدار می‌شوند، تراکم اسپایک‌ها در خروجی رمزنگاری کاسته می‌شود و این اسپایک‌ها دارای بی‌نظمی بیشتری هستند. دلیل شفافیت کمتر واضح است، چون پیکسل‌های بیشتری با هم مورد بررسی قرار می‌گیرند، هر پیکسل سهم کمتری در خروجی ایفا می‌کند. دلیل موج‌های دوم و سوم در شکل فیلتر نیز مستقیما به فرمول مورد استفاده وابسته است. دلیل تراکم کمتر در خروجی رمزنگار نیز آن است که الگوی مورد بررسی در کرنل با ابعاد بزرگ‌تر کمیاب‌تر بوده و درنتیجه فرکانس اسپایک کاهش پیدا می‌کند. بی‌نظمی بیشتر نیز دلالت بر آن دارد که بخش‌های ساده تصویر نادیده گرفته می‌شوند.

B.2. اثر orientation

In [56]:
```
i_max = 18
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
```

```
  simulate(p, encoder='t2fs', name=f'orientation: {i}pi/18', filter_name='Gab
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=9, wavelength=5, std=2, orientation=torch.tensor(i*pi/18), as
p.show()
```

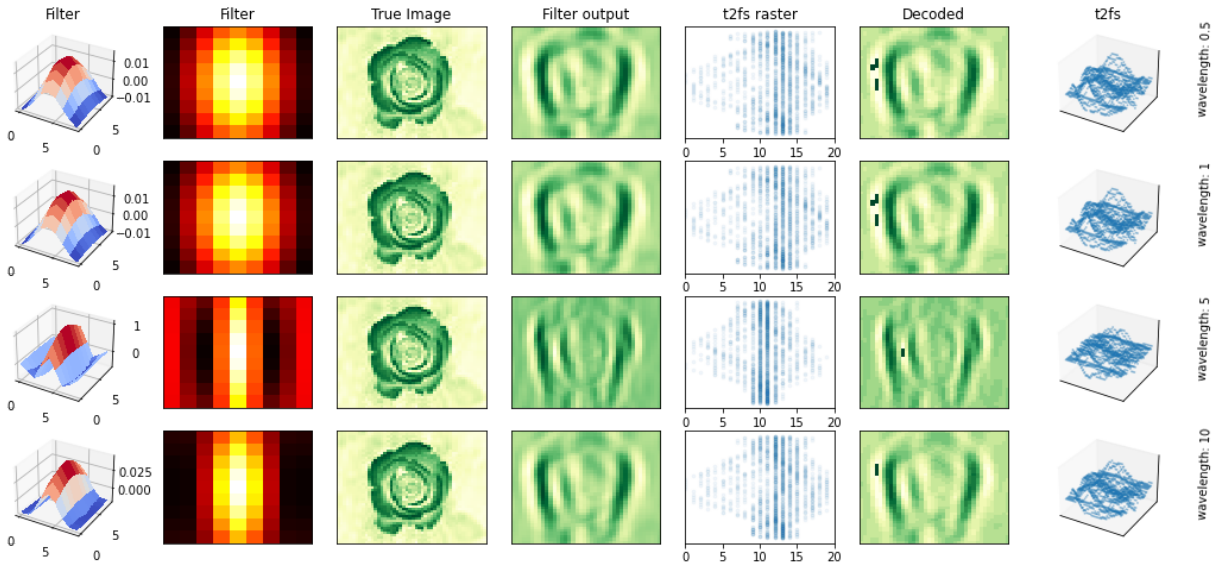




مشاهده می‌کنیم که با چرخش فیلتر (ده درجه به ده درجه)، خطوط با همان درجه در تصویر توسط فیلتر مورد توجه قرار گرفته‌اند. تصاویر مشاهده شده مشابه تصاویر پسر بچه در اسلایدهای درس می‌باشد که حاکی از عملکرد درست فیلتر می‌باشد. توجه به رستر پلات مربوط به رمزنگار نیز خالی از لطف نیست. در هر زاویه‌ای نورون‌هایی که سریع اسپایک می‌زنند متفاوت‌اند که مطابق با خروجی فیلتر است.

C.2. اثر wavelength

In [57]:
```
i_max = 4
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  wavelength = [0.5,1,5,10][i]
  simulate(p, encoder='t2fs', name=f'wavelength: {wavelength}', filter_name='
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=9, wavelength=wavelength, std=2, orientation=torch.tensor(0.)
p.show()
```

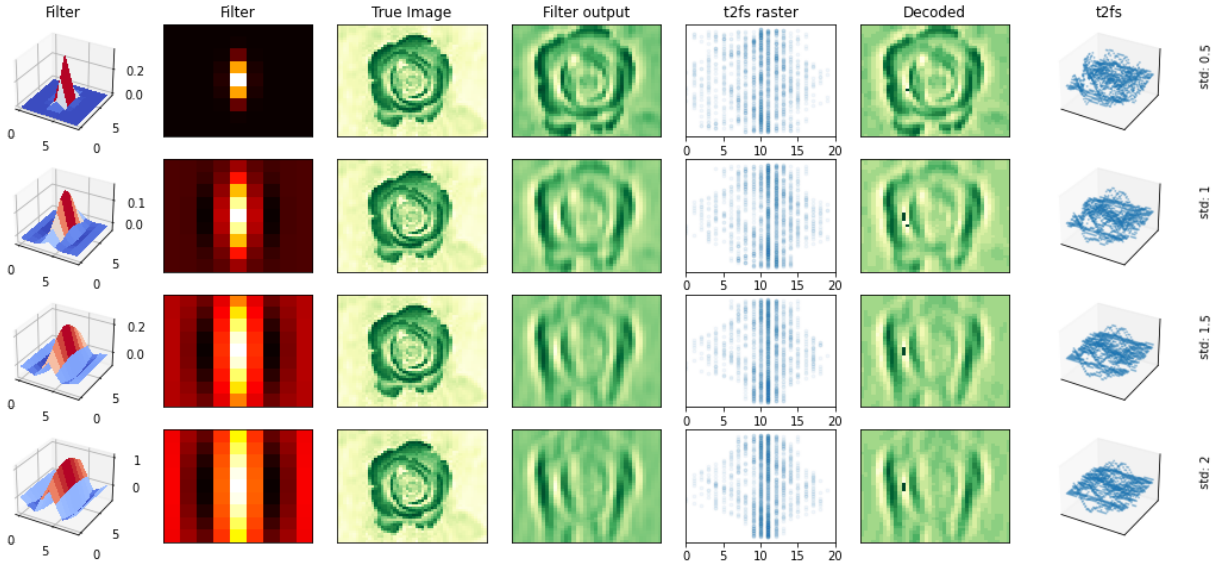


می‌بینیم که این پارامتر کنترل میزان باز یا بسته بودن فیلتر را در اختیار دارد. با افزایش این پارامتر، سرعت نزول مقادیر فیلتر بیشتر می‌شود و در نتیجه خطوط باریک‌تر بیشتر مورد توجه قرار می‌گیرند.

D.2. اثر انحراف معیار

In [64]:
```
i_max = 4
plt.figure(figsize=(18,2*i_max))
p = Plotter([
```

```
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  std = [0.5,1,1.5,2][i]
  simulate(p, encoder='t2fs', name=f'std: {std}', filter_name='Gabor', postfi
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=9, wavelength=5, std=std, orientation=torch.tensor(0.), aspec
p.show()
```

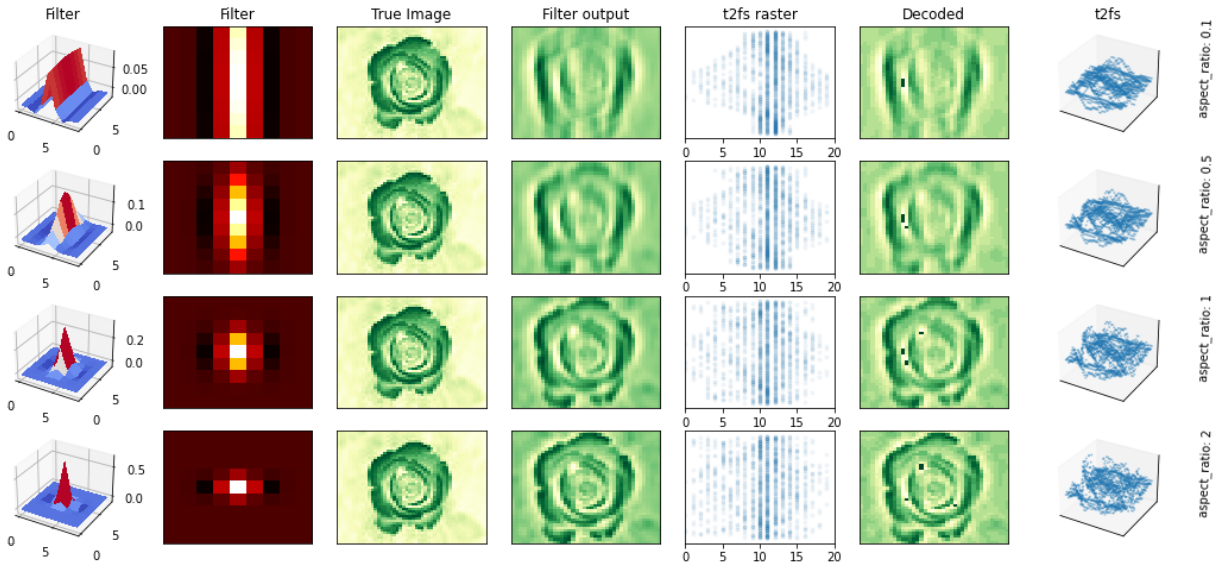


به صورت کلی، با کاهش واریانس، فیلتر متمرکزتر شده و به سمت شکل ضربه میل می‌کند که به معنی تشخیص یک نقطه روشن در زمینه تاریک است. به همین دلیل شکل را به شکل دقیق‌تری بازیابی می‌کند اما ارزش اطلاعاتی برای ما ندارد چون قادر به استخراج خطوط نیست. اگر این پارامتر بیش از اندازه بزرگ انتخاب شود نیز به دلیل پراکندگی بیش از اندازه، باز هم قابل استفاده نخواهد بود.

E.2. اثر aspect_ratio

In [67]:

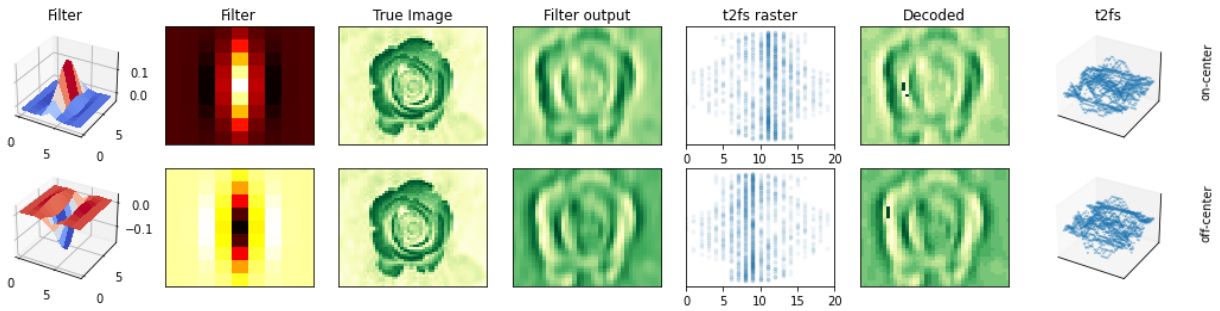
```
i_max = 4
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  aspect_ratio = [.1,.5,1,2][i]
  simulate(p, encoder='t2fs', name=f'aspect_ratio: {aspect_ratio}', filter_na
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=9, wavelength=5, std=1., orientation=torch.tensor(0.), aspect
p.show()
```



همانطور که در آزمایش بالا مشخص است و از پیش می‌دانیم، این پارامتر وظیفه تعیین میزان کشیدگی فیلتر را بر عهده دارد. با مقدار بزرگ این پارامتر، کشیدگی فیلتر کم شده و به سمت فیلتر نقطه میل می‌کند که مطلوب استفاده از این فیلتر نیست. با کاهش بیش از اندازه این پارامتر نیز خطوط بسیار بلند مورد بررسی قرار خواهند گرفت و بسیاری از خطوط که طول بسیار زیادی ندارند نادیده گرفته می‌شوند که ممکن است مطلوب نباشد. اندازه این پارامتر باید بر مبنای تسک مورد بحث انتخاب شود.

F. Off-Center.2

In [69]:
```
i_max = 2
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  simulate(p, encoder='t2fs', name=['on-center','off-center'][i], filter_name
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=9, wavelength=5, std=1., orientation=torch.tensor(0.), aspect
p.show()
```

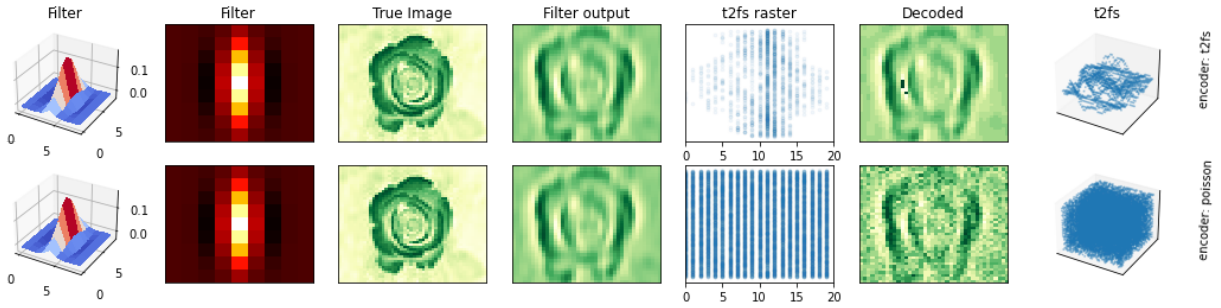


به سادگی تئوری مطرح شده در مباحث درس برای این قسمت، در نمودارها قابل مشاهده است. در فیلتر on-center، خطوط روشن در زمینه تاریک مورد توجه قرار گرفته و روشن‌تر شده‌اند و در فیلتر off-center برعکس. برای صحت‌سنجی این صحبت، به مرکز گل توجه کنید.

G.2. تعامل با رمزنگار time to first spike و پواسون

In [71]:
```
i_max = 2
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
```

```
], wspace=0.17, hspace=0.2)

for i in range(i_max):
  enc = ['t2fs','poisson'][i]
  simulate(p, encoder=enc, name=f'encoder: {enc}', filter_name='Gabor', postf
    transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze
    kernel_size=9, wavelength=5, std=1., orientation=torch.tensor(0.), aspect
p.show()
```

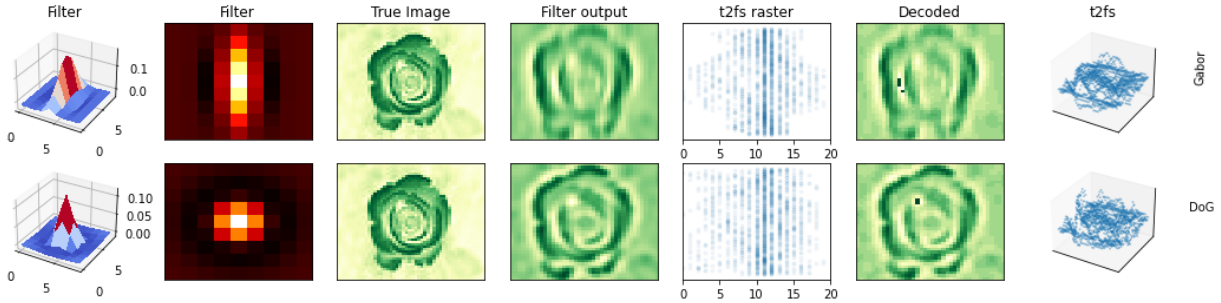


پیش‌تر نیز به اثر فیلترها بر روند کدگذاری اشاره شد. با فیلتر کردن بخشی از دادگان تصویر حذف می‌شود (فقط نقاطی باقی می‌مانند که با فیلتر همخوانی داشته باشند) درنتیجه خروجی رمزنگار تراکم کمتری خواهد داشت و پس از کدگشایی، به تصویر پس از فیلتر تبدیل خواهد شد و توانایی بازسازی تصویر اصلی را ندارد. تفاوت دو فیلتر در اینجا نیز همانیست که در فازهای قبلی بیان شده است.

3. جمع بندی فیلتر DoG و Gabor

پیش‌تر، با استناد به مطالب تدریسی، می‌دانیم که فیلتر DoG برای تشخیص نقاط و فیلتر Gabor برای تشخیص خطوط به کار می‌روند.

In [72]:
```
i_max = 2
plt.figure(figsize=(18,2*i_max))
p = Plotter([
  [f'filter{i}3D',f'heat_filter{i}',f'true_image{i}',f'filter_output{i}',f'ra
  for i in range(i_max)
], wspace=0.17, hspace=0.2)

simulate(p, encoder='t2fs', name='Gabor', filter_name='Gabor', postfix=str(0)
  transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze_(
  kernel_size=9, wavelength=5, std=1., orientation=torch.tensor(0.), aspect_r

simulate(p, encoder='t2fs', name='DoG', filter_name='DoG', postfix=str(1), ti
  transform=transforms.Compose([transforms.ToTensor(),lambda x: x.unsqueeze_(
    kernel_size=9, std1=1., std2=2.)

p.show()
```



در تصاویر خروجی فیلترها در بالا مشاهده می‌کنیم که تصویر حاصل از فیلتر Gabor، حالت کشیده دارد درخالی که تصویر خروجی DoG این چنینی نیست و نقاط از هم مجزا شده‌اند.